Special Purpose, but Domain Independent, Inference Mechanisms

SPECIAL PURPOSE, BUT DOMAIN INDEPENDENT, INFERENCE MECHANISMS

Alan Bundy, Lawrence Byrd and
Chris Mellish

D.A.I. RESEARCH PAPER NO. 179

Paper presented at *European Conference on Artificial Intelligence*, Paris, July, 1982.

SPECIAL PURPOSE, BUT DOMAIN INDEPENDENT, INFERENCE MECHANISMS

by
Alan Bundy, Lawrence Byrd and Chris Mellish

Abstract

We describe a number of special purpose, but domain independent, inference mechanisms. While these mechanisms are limited to certain kinds of inference and inference rules, they do not rely on special properties of the domain, but on logical properties of predicates and rules, which make them equally applicable to other domains. These logical properties include: transitivity, functionality and unarity. The union of these mechanisms handles nearly all the inference required in the Mecho project for solving Mechanics problems stated in English.

Keywords

Inference, Combinatorial explosion, Meta-Level Reasoning, Problem Solving, Mechanics.

Acknowledgements

We thanks the many people who have worked on the Mecho project and who contributed to the ideas in this paper, namely George Luger, Rob Milne, Martha Palmer and Bob Welham. The work was supported by SRC grant GR/A/57954.

1. Introduction

Many problems studied within Artificial Intelligence call for inference. Unfortunately, the general purpose inference mechanisms, which have been built in Artificial Intelligence, are known to be susceptible to combinatorial explosions. One solution to this problem is to build domain specific inference mechanisms. Building such a mechanism is non-trivial and, by definition, the researcher can inherit little from those which have been built before.

This paper takes a different approach: the attempt to build special purpose, but domain independent, inference mechanisms. The mechanisms we will present rely on special properties of the inference rules, predicates, etc that they manipulate. However, these special properties are logical properties, like transitivity, functionality, unarity, etc, and so the mechanisms may be useful to researchers in other domains.

These inference mechanisms are the product of the Mecho research project, [Bundy et al 79], to build a computer program which can solve Mechanics problems stated in English. Mecho uses inference for a number of different purposes: interpreting the meaning of English statements; forming a mathematical model and solving the specified problem; and solving the resulting simultaneous equations. Each time we were confronted with a combinatorial explosion we investigated the nature of the inference rules, predicates, etc, involved, to see how the explosion might be avoided. The range of the problems being solved by Mecho prohibited ad hoc, patch-type solutions. Those solutions which emerged often depended on logical, rather than domain specific properties to do with physics, geometry, etc. At the close of the project most of the Mecho inference rules were catered for by one or other of the special purpose mechanisms. Thus, whilst each mechanism accounts for only a part of the inference being done, the set of mechanisms covers nearly all the inference required by Mecho.

This paper describes the Mecho special purpose inference mechanisms, and outlines the framework into which they fit. The framework consists of meta-level rules which are responsible for allocating inference goals to the appropriate mechanism. It does this by inferring the logical properties of the goals from meta-level assertions of the properties of their syntactic parts (e.g. predicates, arguments, etc).

2. Controlled Term Creation

Perhaps the major cause of the combinatorial explosion in automatic inference systems, is the substitution of terms for variables in formulae. In classical predicate logic, the ability to do this is represented by the substitution rule of inference:

$$\frac{F(X)}{F(t)}$$

where F is a formula, X a variable and t a term. This rule can cause a infinite branch in the search space! In many systems, for example resolution systems, this rule is embedded in the standard rule of inference. In such systems, some of its power is controlled by limiting substitutions so that only the "smallest" instantiation required for some matching task is actually used (this is a property of unification). But these are not just difficulties with resolution, the general problem applies to many different inference systems - indeed any system which allows the introduction of function-like expressions during its inference. Even when the substitutions are limited this rule is too powerful in many practical applications. It causes the introduction of terms which have not previously been encountered in the seach space, many of which are not required in the proof.

For instance, consider the constant acceleration formula:

$$V = U + A.T$$

where U and V are the initial and final velocities of an object, A its acceleration and T the duration of the period of travel.* The implication of the words following 'where' above are that V, U, A and T are functions of the object, O, and the period of travel, P, i.e.

$$v(O,P) = u(O,P) + a(O,P).t(P) \qquad \text{(i)}$$

Suppose that in a Mechanics problem about a car, the initial velocity, acceleration and distance travelled, are given, but the final velocity is sought. Application of formula (i), say as a rewrite rule, to find the value of v(car,trip), will unify O to car and P to trip and instantiate t(P) to t(trip). But the duration of the trip is not given, and may not be easily calculated, leading to a long search and a non-optimal solution.

An experienced Mechanics problem solver would not use formula (i), but formula (ii) instead.

$$v(O,P)^2 = u(O,P)^2 + 2.a(O,P).s(O,P) \qquad \text{(ii)}$$

where s(O,P) is the distance travelled by O during P. Since s(car,trip) is already given, no new terms are introduced into the search space. In Mechanics terminology, this is known as not introducing unnecessary intermediate unknowns.

Thus the important difference between these two problem solving strategies lies in whether or not certain additional terms are introduced during the inference. We have devised a term creation mechanism in which the introduction of the intermediate unknown, t(trip), is controlled by problem specific considerations. The instantiation of t(P) to t(trip) by unification, makes implicit use of the existence property of functions, that is, given P, the quantity t(P) is guaranteed to exist. Our term creation mechanism works by removing the existence property, so that unification cannot use it willy nilly, and replacing it by an explicit creation rule of inference, which is only applied in suitable circumstances.

---

*Throughout this paper we will follow the convention that variables always start with an upper-case letter, while constants start with a lower-case letter.

The existence property is removed by the replacement of functions by relations. For instance, formula (i) is rewritten as:

initvel(O,U,P) & finvel(O,V,P) & accel(O,A,P) & duration(T,P)
--> V = U + A.T

Now if the value of v is sought, where finvel(car,v,trip), then various subgoals will be set up, namely, finding the value of U + A.T where initvel(car,U,trip), accel(car,A,trip) and duration(T,trip). Note that T is not automatically instantiated, i.e. no intermediate unknown is introduced.

The existence property is reintroduced by an explicit creation inference rule:

$\forall$ X,Z $\exists$Y r(X,Y,Z)
---
r(s,f(s,t),t)

where r is a predicate, X and Z are vectors of variables, Y a variable, s and t are vectors of terms, f a new skolem function, and r(X,Y,Z) has the existence property for Y. This would be as explosive as the substitution rule, if used forwards in an undirected way. Therefore, it is usually used backwards to satisfy a subgoal of the form, r(s,Y,t), by substituting f(s,t) for Y.

The version of this rule in Mecho was only designed to handle the case where s and t are ground terms. In this case f might as well be a nullary skolem function (i.e. just a new gensymed constant), as its dependancy on s and t provide no new information. Because of this, the backwards use of the creation rule does not generate any search, but satisfies the subgoal in a single step

The hypothesis part of the rule is stored, not as an object level, predicate calculus formula, but as a piece of meta-information about the predicate r, stored in the form,

exists(r(X,Y,Z),X u Z)

This assertion is used by the (meta-level) inference system to justify the use of the explicit creation rule for the predicate r. Note that r does not have to be a function from X and Z to Y, because the Y need not be unique. Thus the creation rule could also be applied to the relation, parent(X,Y), meaning Y is the parent of X. Every person has two parents, so the predicate has the existence property without the uniqueness property. Separating the two properties of functions, existence and uniqueness, thus allows predicates which only have one of these properties to be uniformally handled, as well as making clear the different inference steps that these properties justify. Uses of uniqueness properties are described in sections 5 and 6.

Note also that a relation can have existence and/or uniqueness properties in more than one way, and so a single relation can be made to replace two or more functions, provided suitable meta-properties about it are stored. For instance, the relation time-sys(Period,Initial,Final), meaning Initial and Final are the first and last moments of a period of time, respectively, can replace the functions, initial(Period), final(Period) and period(Initial,Final).

A first order theory in which all functions have been replaced by relations has only a finite Herbrand Universe, and hence generates only finite search spaces. This fact has been used, with great effect, to eliminate the combinatorial explosion in automatic inference, by [Bundy 73, Gelernter 63, Nevins 75, Wos et al 65]. The use of the creation rule restores the infinite search space, but its controlled use provides an effective way of guiding the search.

For example, in [Gelernter 63], the creation rule corresponds to the construction of new points as the intersection of non-parallel lines. The Geometry Machine explored the entire finite search space, using only those points mentioned in the original diagram. It then picked a subgoal which was not true in the current diagram, but could be made true in a diagram extended by one point. This point was

constructed, thus extending the space, and the whole process was then applied recursively.

Mecho uses essentially the same idea, when forming equations. It tries to solve for an unknown using equations containing only this unknown and quantities given in the problem statement. If this fails it uses one of the other equations, invoking the creation rule to introduce the required intermediate unknowns. These intermediate unknowns must then be solved for in turn, but again the creation rule is only invoked when attempts to proceed without it have failed.

The control regime used by both these programs is:

. Explore the entire finite space, without using the creation rule.

2. If the problem is still not solved, then pick an unsatisfied subgoal in this space and use a backwards application of the creation rule to satisfy it.

3. Apply the same control regime to the extended space.

This regime allows the gradual development of an infinite space by the recursive development of a nested series of finite spaces. Although quite successful as it stands, its success can be enhanced by the careful choice of the unsatisfied subgoal in step 2.

For more details on controlled term creation see [Bundy 77]

3. Equivalence and Similarity Classes

Another major cause of combinatorial explosions in automatic inference are the symmetric and transitive laws, e.g.

```
Symmetry:      X=Y --> Y=X
        and
Transitivity: X=Y & Y=Z --> X=Z
```

These laws combine with themselves and each other to produce infinite chains of redundant inferences. This has long been one of the acknowledged difficulties of incorporating the = relation into an automatic inference system, but our research has revealed that the same difficulty obtains for a wide class of other relations, for instance, the relation, sameplace(X,Y,T), that two point objects, X and Y, are at the same place at some time, T. In this case the laws are:

```
Quasi-Symmetry:      sameplace(X,Y,T) --> sameplace(Y,X,T)
        and
Quasi-Transitivity: sameplace(X,Y,T) & sameplace(Y,Z,T)
                                  --> sameplace(X,Z,T)
```

In the case of equality, various solutions to this problem have been adopted. These include the use of equality classes [Nevins 75] and the use of systems of rewrite rules [Huet 77]. We have adapted these solutions to deal with relations, like sameplace, which obey quasi-transitive, quasi-symmetric and also quasi-reflexive laws;

```
Quasi-Reflexivity:  sameplace(Y,X,T)
```

We have replaced the conventional axiomatization of these predicates by an axiomatization which does not have the same explosive properties as the standard axioms given above. We will see that this axiomatization incorporates the intuitive notions of the equivalence class and rewrite rule solutions.

Our solution consists of a general axiom scheme, with which quasi-equality relations can be defined. This scheme contains the extra argument positions, like the time argument of sameplace, which are needed by the defined relation. Any

relation noted as having quasi-equality properties is automatically defined using this scheme.

This general axiom scheme is:

Rootof(X,Root,Aux) & Rootof(Y,Root,Aux) --> Reln(X,Y,Aux) (iii)

Arc(X,Next,Aux) & Rootof(Next,Root,Aux) --> Rootof(X,Root,Aux)

not ∃Next Arc(X,Next,Aux) --> Rootof(X,X,Aux) (v)

To make this into a set of axioms defining a particular predicate, say sameplace, we must instantiate Reln to sameplace, and Rootof and Arc to new relation names, say rootof1 and arc1. Aux will play the role of the time argument of sameplace. In the general case where the defined predicate has more than one auxiliary argument, Aux will be a set.

These axioms can best be understood as describing properties of a set of trees, where the nodes are labelled with names of entities and the arcs represent Arc relations between them. For instance, the arc1 relations

arc1(fred,trainA,then)
arc1(trainA,station1,then)
arc1(trainB,station1,then)
arc1(fred,trainB,now)
arc1(trainB,station2,now)
arc1(trainA,station3,now)
arc1(doris,station3,now)

can be understood as describing the trees in figure 3-1.

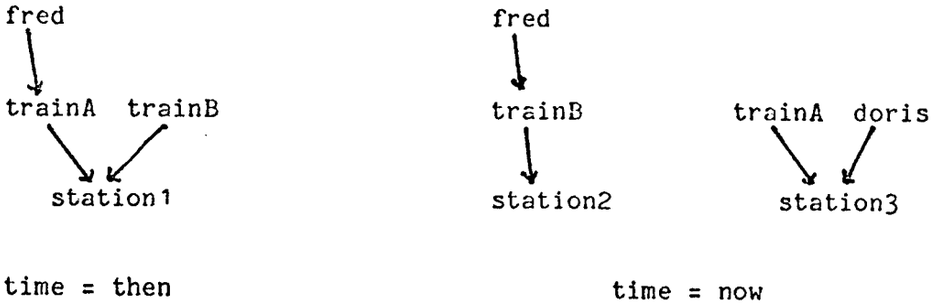

Figure 3-1: Sameplace Trees for Then and Now

The idea is that for each Aux argument there is a set of trees: two objects, X and Y, being in the same Aux tree iff Reln(X,Y,Aux). Two objects, X and Y, are joined by an Aux arc directed from X to Y iff Arc(X,Y,Aux). The root of each tree is a distinguished element: Root being the root of an Aux tree containing X iff Rootof(X,Root,Aux). This assumes that the Arc relations are all ground units and that for each Aux and X there is at most one Y such that Arc(X,Y,Aux). The inference mechanism is responsible for seeing that this is so.

Axiom (iii) says that two objects are in the relation if they have the same root, i.e. are in the same tree. Mecho uses this axiom backwards, to show that two objects are in the relation. Axiom (iv) says that the root of the successor of a node is also the root of that node. Axiom (v) says that a node with no successors is its own root. Mecho uses axioms (iv) and (v) backwards, to find recursively the root of an object or to determine whether the root is a given node.

These trees can be viewed as equivalence classes, where the root node is a representative of the class. The mechanism, defined by Mecho's backwards use of axioms (iii), (iv) and (v), tests that two objects are in a relation by checking that they are in the same equivalence class. It does this by checking that they share the same class representative. For this reason we call this special purpose inference mechanism, the equivalence class mechanism.

The trees can also be viewed as the derivation trees of a canonical form

mechanism, where each arc represents a rewriting of one object into another, and a root node represents the canonical form of its predecessors. Under this view, the equivalence class mechanism tests that two objects are in a relation by rewriting each into its canonical form and then checking these for identity. Currently, these two views of the mechanism are isomorphic, but we expect the rewrite rule view to be more suggestive of a solution to the problem of representing additional axioms about quasi-equality relations, other than ground unit clauses, like those in figure 3-1.

The class of relations exhibiting quasi-equality, explosive properties also includes relations like rel-vel(p1,p2,v,t), that two objects, p1 and p2, have a relative velocity, v, at time t.

```
Quasi-Symmetry:      rel-vel(X,Y,V,T) & inverse(V,V')
                                  --> rel-vel(Y,X,V',T)
Quasi-Transitivity: rel-vel(X,Y,V1,T) & rel-vel(Y,Z,V2,T) & add(V1,V2,V)
                                  --> rel-vel(X,Z,V,T)
Quasi-Reflexivity:  rel-vel(X,X,zero,T)
```

Where zero is the vector with 0 magnitude, inverse calculates vector negation and add is vector addition. Unlike the time argument of the sameplace predicate, the velocity argument of rel-vel, is not a passive passenger, but has different values in different occurrences of the rel-vel relation. The equivalence class mechanism can be extended to handle such arguments by making the arguments label the arcs of the trees, as pictured in figure 3-2.

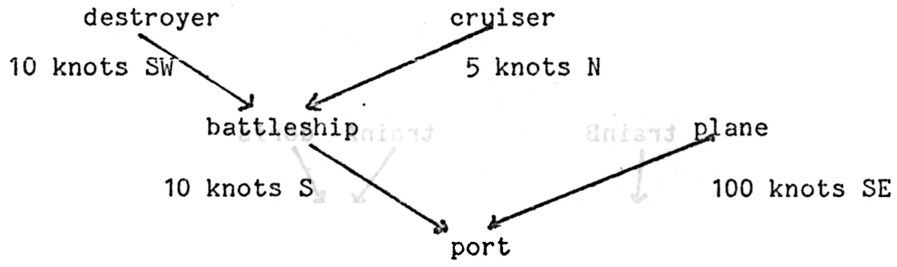

Figure 3-2: Relative Velocity Similarity Tree

The predicates Reln, Rootof and Arc must be given an extra argument, called the arc label argument. Axioms (iii) and (iv) must be modified to include additional conditions to make the necessary arc label combining calculations. We call the extended mechanism the similarity class mechanism.

The similarity mechanism can be used, for instance, to calculate the relative velocity between the plane and the cruiser in figure 3-2. The shared root of each, port, is found and the relative velocity of each to the port is calculated by vector addition of the velocities labelling each arc of the route. The relative velocity of the cruiser to the port is then subtracted from the relative velocity of the plane to the port to give the relative velocity of the plane to the cruiser.

In fact, the similarity class axioms are not a simple modification of the equivalence class ones, since care has been taken to return the simplest possible arc label combination (e.g. velocity vector). This is done in two ways: by trying to preserve the Reln arc labels as Arc arc labels when building trees; and by finding the shortest path between two nodes in a tree, rather than going to the root each time. This latter modification makes the similiarity class axioms rather complicated, and for this reason we omit them here.

Our main difficulty with both the quasi-equality and similarity class mechanisms has been the question of how to introduce extra axioms, apart from the ones described above, while maintaining the efficient inference properties of these mechansims. In practice this has not been too much of a problem within Mecho, and we have handled the few cases that arose using forward inference rules that generate

additional facts from the extra axioms, which are then added to the class trees in the usual way. However, we are aware that this area requires further investigation. For more details of the equivalence and similarity class mechanisms see [Bundy 78].

4. Inference Involving Unary Predicates

Another place where conventional inference techniques can get into trouble concerns axiomatisations involving unary predicates. Consider a situation where there is a large number of axioms of the form:

```
particle(X) --> physobj(X)                                   (A)
particle(X) --> point_shape(X)                               (B)
rod(X) --> physobj(X)                                        (C)
man(X) --> particle(X)                                       (D)
man(X) --> male(X)                                           (E)
......
man(john)
rod(rod1)
woman(mary)
......
```

and inference goals like

```
physobj(john)
```

When such a goal is encountered, there are two standard possibilities. Inference can proceed forwards from the axiom 'man(john)' (deriving 'particle(john)' and so on until the goal is encountered). Or inference can proceed backwards from the goal (generating subgoals 'particle(john)' and so on until an axiom is encountered). However we do it, there are two basic problems. First of all, the chains required may be quite long (man-particle-physobj here, but longer chains are often encountered). Secondly, there may be a choice at each point which axiom to use. For instance, going forwards, should we use (D) or (E)? If (D), should we then use (A) or (B)? Because these choices are multiplied for each step of the chain, we have quickly run into a combinatorial explosion.

Our solution to this problem has been to devise a special purpose inference mechanism for unary predicates. By suitably restricting the kinds of inference goals that may be encountered and the kinds of general rule that may be expressed, we have reduced the problem to one that is amenable to an efficient solution with no search. We follow Hayes and Hendrix in viewing the well known notion of a type as corresponding directly to the use of unary predicates in logic. Thus, our special inference mechanism is really just a form of type checking, where the inference rules express certain restricted relations between types. The type checking is implemented using the operation of unification of patterns representing types (described below).

4.1. Unary Predicates as Types

In Mecho, inference involving all unary predicates is handled by this special type checking mechanism. All the objects in a problem are typed and these types are taken to apply over the whole problem. In our representation time arguments are always explicit; any property that may only be valid for part of the time will be represented by at least a binary predicate, and so we are free to assume that unary predicates only express certain kinds of constant property.

Given the identification of unary predicates with essential properties and types, we have been able to restrict the kinds of general rules involving unary predicates. As a result we did not need the full power of logical inference; instead, a special purpose mechanism was adequate. The following are our principle restrictions:

- Within our system there are only two kinds of inference goal that may arise for unary predicates. Firstly, one may ask whether a particular object has a particular type. Second, one may ask whether there exists an object with a given type or what objects are currently known to have a

given type. In fact, we have found goals of only the first type quite adequate for most of our purposes. The second kind of goal might arise in natural langauge understanding, where the referent of a phrase like "the particle" is to be determined. However, given some way of enumerating the objects in current "focus" [Grosz 77, Sidner 79], we can easily reduce the problem to that of determining whether a particular object has the type. Thus our implementation optimises type checking while leaving the generation of objects of a type comparatively expensive.

Since unary predicates represent essential properties, an inference rule that enables one to prove that a type holds of an individual need not involve any reference to other individuals or involve anything other than other properties of the same individual.

We will also make the simplifying assumption, for this section, that for every individual, there is given a conjunction of predicates that exhaustively describes what kind of object it is. Of course, this does not mean that ascertaining whether an arbitary property holds will always be trivial. In section 4.3, we will see that our type mechanism can actually handle incompletely described objects as well.

These restrictions led us to consider an inference system of a restricted kind as we now explain.

First of all, the main kind of rule that needed to be expressed was one that expressed how the world breaks down into parts, and how those parts themselves decompose. We chose a reserved predicate 'entity' to represent the property that holds of all objects, and disjoint union of sets as the primitive operation for description. In the logic, this is treated by the use of a variadic connective oneof, which asserts that exactly one of a set of propositions is true. Thus one of our rules is:

entity(X) <--> oneof {male(X),female(X),neuter(X)}

This expresses one classification of the objects in the world. Another independent classification is given by:

entity(X <--> oneof {zero_d(X),one_d(X),two_d(X),shapeless(X)

Given these rules, any object in the world can be fully described by two properties, corresponding to its gender and its "dimensionality". If there were rules describing how 'one_d' objects decomposed, the description would, of course, have to include a mention of how the object behaved relative to that sub-classification. If we wanted to refer to some particular combination of types more briefly, we can do so by using another kind of rule involving conjunction. For instance:

man(X) <--> zero_d(X) & male(X)

might be appropriate for the idealised view of men taken in most mechanics problems. Predicates introduced in these conjunctive rules can themselves be further described by 'oneof' rules, for instance:

man(X) <--> oneof {boy(X ,bachelor(X

To summarise, the type checking system incorporates a reserved predicate 'entity' and the possibility of rules of two types, as exemplified above. In addition we make the extra stipulation that every unary predicate (except 'entity') appears exactly once either on the "left hand side" of an conjunctive rule or on the "right hand side" of a 'oneof' rule. We are hence dealing with a considerably restricted subset of the unary Predicate Calculus. Indeed, it is in some sense even more restricted in power than the Propositional Calculus. The two kinds of rules form the basis of an efficient special purpose inference mechanism. This mechanism is capable of answering questions about whether particular objects have particular properties,

given arbitrary complete descriptions of the objects as conjunctions of simple type assertions.

4.2. Representing Types as Logical Terms

The fundamental idea behind the implementation of this inference mechanism is that of representing each type property by a pattern (a term of logic), such that the intersection of two properties corresponds to the pattern which is the most general unifier of the terms associated with the two properties. Moreover, two terms will fail to unify iff the corresponding properties are incompatible. For instance, assuming the inference rules given above, we might have the following property-term associations:

```
entity          --      entity(_,_)
male            --      entity(male(_),_)
female          --      entity(female,_)
neuter          --      entity(neuter,_)
zero_d          --      entity(_,zero_d)
one_d           --      entity(_,one_d)
two_d           --      entity(_,two_d)
shapeless       --      entity(_,shapeless)
man             --      entity(male(_),zero_d)
boy             --      entity(male(boy),zero_d)
bachelor        --      entity(male(bachelor),zero_d)
```

where the isolated underline symbols represent variables that only occur once. Given these basic patterns, we can construct new patterns for conjunctions, e.g.:

```
one_d female            entity(female,one_d)
```

But not all combinations are possible. A "one_d bachelor" will be disallowed because

```
entity(_,one_d) and entity(male(bachelor),zero_d)
```

do not match.

The idea of representing types by logical terms has been used before in other contexts, for instance in [Dahl 77]. One innovation here is the fact that we automatically generate the logical terms in accordance with the supplied relationships between the properties, i.e. the unary predicate axioms.

In addition to keeping a logical term for each property, it is also necessary to keep a term for each known object. This expresses the total type knowledge about the object. To determine whether an object has a given property, it is then only necessary to see whether the type pattern of the object unifies with the the type pattern of the property. This is an operation linear in the size of the patterns, and the size of a type pattern is at worst a linear function of the number of 'oneof' rules. Moreover, it involves no search.

4.3. Handling Incomplete Type Information

The inference system for unary predicates can handle situations where there is incomplete information about the type of an object. When the type of an object is not completely known, there are variables in the object's type pattern corresponding to subclassifications where the object has not yet been placed. In this case, checking unification of type patterns corresponds to checking that a type is "compatible" with what is already known.

In the natural language understanding part of Mecho, incomplete type information can arise when the system is handling an unevaluated referent of a phrase. For instance, one would like to perform certain operations on the referent of a phrase like "it" before making a decision about what it is. In fact, noun phrase referents are treated just as variables whose values are to be determined by (possibly postponed) inferences (section 6). Thus the situation of incomplete type information about referents merges into the general situation of incomplete type

information about variables during inference

In order to cope adequately with incompletely described objects, the type checking system has to be extended, so that when an individual's type pattern is unified with the pattern of the type being tested, the result of the unification is stored as the new type pattern for the individual. This means that, having assumed that the individual has the type for some purpose, it is not thereafter able to make a conflicting assumption (except by backtracking to where the first assumption was made). Also, it is useful to have a record of the types that the individual must have in order for the current analysis to be valid. This capability is used in a limited facility for handling unknown nouns and adjectives in Mecho. For instance, in dealing with:

Two particles are connected by a flurgle.

Mecho initially assigns a default type 'entity' to the "flurgle". However, the analysis of the verb "connect" involves making inferences about the object doing the connecting. This results in the flurgle being represented as an object capable of connecting things. In our representation of the Mecho world, the appropriate type is a specialisation of 'physical object' that includes strings and springs.

5. Use of Negative Information

The inference systems in Mecho are principally used to derive positive information from the problem specific information along with general facts and rules about the domain. When, for example, the problem solver is trying to apply a particular problem solving strategy, its goals will be questions about whether certain objects exist in the right relationships, and if so, what properties (masses, velocities etc) they have. We have regarded negative information as a way of pruning the search space. That is to say, when a goal of the form P is set up, checks can be made to see if not P can be derived. If the proof of not P succeeds then obviously the goal P should fail, and no effort should be spent trying to derive P. This stategy will only be effective if proving not P is a considerably simpler task than proving P. In the general case it will be just as easy to get lost in a combinatorial search while attempting to derive not P as P. It has therefore been our intention to associate simple, special purpose, methods of deriving not P with all of our predicates. Such special methods are guaranteed to terminate quickly and can therefore be used as quick checks on the goals generated during proofs.

In section 4 we described a proof mechanism for a restricted class of axioms in unary predicate calculus. If we encode a complete type hierarchy (for the purposes of the program) in this way then we have a decision procedure which allows us to say for any two types whether or not they are compatible.

I.e. if we know solid(obj1) & one_d(obj1) & stretchable(obj1)

then not particle(obj1)

This follows immediately, given the nature of our type axiomatisaton: being a particle is not compatible with being one dimensional. If we were trying to prove 'particle(obj1)' we would thus fail the goal. If our type information about an object is complete, i.e. we know for every division of our universe into disjoint types, exactly which type the object is; then proving the positive case becomes the same as checking the negative case. For such objects all unary predicate goals are fully decidable. However, it is more common to know only the type of an object along some subset of all possible dimensions, and here there will be an intermediate case that what we are trying to prove neither follows from, nor is contradictory with, what we already know. This is the often the case during the natural language analysis where, as described above in section 4.2, the inference system will be gradually improving its incomplete type information. Here the negation check forms part of the merging of type information: impossible type merges being immediately rejected.

This use of type information for pruning impossible goals, can extended to apply to all predicates in our representation. It suffices only to have a record of the type restrictions that the arguments of each predicate must satisfy, for instance an

axiom like:

velocity(X,V,T) --> physobj(X) & velocity(Y) & time(T)

Whenever we are about to attempt a 'velocity' goal, we can then check that the arguments satisfy the type restrictions ("coercing" incomplete type information as necessary). The goal can be rejected if the type checking is unsuccessful.

These examples are unusual in that our tightly controlled use of unary predicate axioms makes attempted proofs of not P very straight-forward. A more general example concerns predicates which have uniqueness properties (cf section 2), which means that goals involving that predicate are guaranteed to have unique solutions, given that certain of the arguments to the goal are known. We represent this information with meta-facts such as:

unique(at(P,Place,Time), {P,Time})

which states that particles, P, can only be 'at' one Place at any given Time. This fact justifies the inference rule:

at(P,Place,Time)
--> ∀ Other (Other ≠ Place --> not at(P,Other,Time))

Let us suppose that we have

Known (in database): at(p1,startpoint,period1)
and
As a goal: Prove at(p1,endpoint,period1)

Given the above inference rule and what is already known about p1, it suffices to know that startpoint and endpoint are different objects for us to immediately derive not at(p1,endpoint,period1) and thereby prune the goal. The inequality of startpoint and endpoint would be established by a use of a unique name assumption in our current implementation. However other, more sophisticated, methods for proving that two objects are different objects could also be used here.

The special method here thus applies to fully ground goals whose predicates have uniqueness properties, and basically consists of checking that we don't already know a fact that is immediately contradictory. Note that the proof of the subgoal at(p1,Place,period1), where Place ≠ endpoint, is deliberately limited to ensure that this check is reasonably cheap. Simply trying to satisfy it by looking in the database in a particularly tight limitation; more generally we can allow proof methods which are known to be inexpensive - such as some of the special mechanisms described in this paper.

In a similar way we can incorporate various restrictions on the arguments of predicates, such as stating that a predicate is aliorelative (which means that the relation can never be used reflexively):

The meta-fact: aliorelative(support(A,B,Time), {A,B})

Justifies the rule: ∀ X not support(X,X,T

Given a goal such as support(block1,block1,period1), it is easy to see that we can immediately derive its negation and thus prune the goal.

In this description, all these special methods have been applied to ground goals (that is, goals that contain no variables). However, when solving general goals for these predicates we can view these quick "negation" checks as constraints on the final values of the variables involved, which can be associated with the goal and checked whenever the appropriate variables become instantiated. Such an approach has been successfully used in parts of Hecho.

It is important, perhaps, to point out that "silly" goals continuously arise within the problem solver and the natural language understander; for example, when checking candidates for some problem solving method or trying to satisfy some unsatisfied reference (see also section 6). It is essential that these cases are handled quickly in a uniform manner.

6. Dynamic Ordering of Inference Goals

This section discusses, not so much a special purpose inference mechanism for a class of predicates as an optimisation of standard inference that makes use of certain properties of predicates. The optimisation concerns the order in which inference goals are attempted, and is similar to techniques used by Pereira and Porto [Pereira and Porto 80] in logic programming.

Consider the following inference rule expressing what it is to be stationary:

stationary(X,T) <--> velocity(X,Y,T) & magnitude(Y,Z) & measure(Z,0,ft/sec)

This says that X is stationary at time T precisely when it has a velocity vector Y whose magnitude is Z and Z measures 0 ft/sec.* Say we need to find out whether 'particle1' is stationary at time 'moment1'. Using this rule, we have to find values for Y and Z such that three propositions are true. We might then treat these three as inference subgoals. The question is: in what order should we attack them? The goals initially look like:

velocity(particle1,Y,moment1)
magnitude(Y,Z)
measure(Z,0,ft/sec)

Looking at the third goal, we can possibly find many quantities Z that measure 0 ft/sec. Therefore, tackling this subgoal first might introduce some search among possible values for Z. Similarly, tackling the second goal first would involve searching among all the quantities that have magnitudes - which may be a number of possibilities. On the other hand, the first goal can only succeed in at most one way. This is because a given object can only have at most one velocity at any given time - something we know in general about the 'velocity' predicate. So it makes sense to tackle the first goal first. When it has succeeded, a value ('velocity1', say) will have been obtained for Y, and the following subgoals will remain:

magnitude(velocity1,Z)
measure(Z,0,ft/sec)

We can pick the next subgoal to be attempted by the same criteria. Any vector has at most one magnitude, and hence the first of these can only succeed in one way. On the other hand, the second goal may be solvable in any number of irrelevant ways.

In Mecho, we have experimented successfully with a very simple mechanism for dynamically determining the order in which inference subgoals should be attempted. This mechanism involves considering the goals individually (rather than globally) postponing those goals that do not have guaranteed uniqueness properties. As we saw in the above example, a goal that is postponed can later obtain uniqueness properties when a variable becomes instantiated by the solution of some other goal. At this point it can be sensibly tackled, even though it was previously postponed. Therefore postponed inference goals must be stored in association with their unbound variables, in order that they can be reconsidered if a value suddenly becomes known.

*In some parts of Mecho, we need to have symbols denoting both the speed and the velocity of an object (so that we can specify that either are 'given' or to be 'sought'). We also need to be able to talk about a quantity without knowing what it measures. Hence the need for the predicates 'magnitude' and 'measure'. It simply would not be adequate to say 'velocity(X,0,ft/sec,T)'

There must also be a default strategy, for resolving the situation when all current goals have been suspended because they all lack the appropriate properties.

Our approach to goal selection differs from many other approaches in Logic Programming and Theorem proving in that our criteria for ordering goals is based on semantically significant (meta-level) properties of the predicates involved, and not ad-hoc syntactic criteria.

In the above example, the order of goals would in fact always be the same whenever the rule was used. This does not always happen. For instance, consider the following rule about what it is to be a left thumb.

leftthumb(X,Y) <--> lefthand(X,Z) & finger(1,Z,Y)

which says that Y is the left thumb of person X if Z is the left hand of X and Y is the first finger of Z. If we had the goal to find a W such that 'leftthumb(john,W)' then uniqueness considerations would suggest tackling the goals in the order written. On the other hand, given the goal to find a person W such that 'leftthumb(W,thumb29)', the other order would be most appropriate. So the dynamic ordering of goals cannot entirely be replaced by adding explicit ordering information to the rules at "compile time".

The selective postponement of non-unique goals provides a kind of solution to the problem of when to perform reference evaluation in natural language understanding. In Mecho, we regard reference evaluation simply as the instantiation of variables subject to constraints. For instance, the referent of the phrase:

the particle of mass 5 lbs

would be something like a variable X, which would eventually have to become instantiated to a value such that:

$\exists$Y particle(X) & mass(X,Y) & measure(Y,5,lbs)

Similarly, the referent of the phrase

it

would initially be a variable, to later become instantiated to an object that has been mentioned in the recent discourse context. As semantic interpretation proceeds, extra constraints are placed apon the referents of phrases. For instance, by the time the sentence:

It connects two particles of mass b and c

has been interpreted, the referent of "it" is also constrained to be something that can connect two objects (a string or spring, say). What form do these constraints take? In Mecho, they take the form of (possibly postponed) inference goals whose solution would cause the instantiation of the appropriate variable. In this context, the problem of when to perform reference evaluation, as discussed by Ritchie [Ritchie 76] and Mellish [Mellish 81], amounts to the question: how many constraints should have accumulated (inference goals been postponed) before a guess is made as to what the referent is (the goals are finally tackled). Clearly, the wrong guess may be made if the goals are tackled too soon. Using the uniqueness properties of goals at least provides some criterion. In particular, Ritchie's example of where too early reference evaluation would be foolish:

The President of US

can be easily handled. Ritchie's point is that, before the appropriate time is established, the number of possible candidates is huge and so evaluation should not be attempted. In our system, such a phrase would give rise to an inference goal: find X and T such that

president(us,X,T)

where X is the "referent" and T is the "current time". Before T was known, the goal

would have no uniqueness properties and would be postponed. However, once T was established (for instance by the reading of a phrase like "in 1962") the goal would be tackled and the reference evaluated.

7. Meta Level Inference

The special inference mechanisms described in this paper are not simply a collection of isolated methods. An important part of our work has been to integrate all these methods into a general approach for inference control, which we outline in this section (see also [Bundy et al 79]).

Central to this organisation is a meta-level database (metabase) containing assertions about the predicates used in the object-level representation (our mechanics domain). These assertions state that the predicates have the various properties described in this paper, such as uniqueness or aliorelativity or being a quasi-equality or similarity relation. Indeed, we regard the whole of the object-level axiomatisation as a series of assertions in the metabase; thus treating inference rules, about mechanics and so on, as meta-level facts that we know apply to the predicate involved. Inference rules themselves can be classified according to the sort of task we wish them to perform; traditionally forward and backward rules are distinguished and we have made use of these as well as normal form rules, prediction rules and others (these are not described in this paper). Such different kinds of rule are stored as meta-level assertions under different meta-level predicates. Thus for each object-level predicate the metabase will contains a range of assertions such as:

```
function(Pred,How)
exists(Pred,How)
unique(Pred,How)
aliorelative(Pred,How)
type_rule(Pred,Rule)
object_level_rule(Pred,Rule)
normal_form(Pred,Rule)
```

These meta-predicates provide us with a vocabulary with which we can describe our object-level axiomatisation. Controlling search at the object-level involves taking advantage of the properties which this vocabulary highlights - by applying special purpose mechanisms, such as we have discussed in this paper, where they are applicable. We implement this as a set of rules which examine goals and decide which properties hold and which methods are applicable. Not only do these rules describe how goals should be solved, but the manipulations brought about by running these rules actually constitute the object-level proof. I.e. our inference mechanism consists of a meta-level axiomatisation over goals; predicates and their properties; proof plans and methods. Our use of meta-rules differs from that of [Davis et al 77] and others, in that we do not just regard them as preference criteria over a separate object-level search space; but rather we use meta-rules to completely specify the whole structure of the search space.

This approach to inference control is a key element in all our work (e.g. [Bundy and Sterling 81]). The use of a declarative meta-level provides a uniform framework within which all the special mechanisms we have introduced can be described and implemented.

8. Conclusion

In this paper we have described several special purpose, but domain independant inference mechanisms. Each mechanism handles a limited class of inferences; exploiting the special, logical, properties of the predicates involved in order to guide search during inference, and to avoid the normal explosive nature of certain kinds of axioms. The mechanisms described are:

- Controlled term creation: which avoids the explosive nature of the uncontrolled introduction of new terms during unification by isolating the

existence properties of functions in a controllable rule of inference

Equivalence and similarity classes: which avoid the explosive nature of quasi-symmetry and quasi-transitivity, by building these properties into the quasi-equality testing and recording mechanisms.

- Type handling: which avoids the possibly explosive nature of type heirarchies by restricting the type axioms allowed so that an efficient inheritance testing mechanism can be used to satisfy all goals involving unary predicates.

Simple negation checking: which prunes the search space by quickly discarding obviously silly goals, making use of properties such as typing, uniqueness and aliorelativity.

- Goal Ordering: which improves the search behaviour through limiting unnecessary backtracking by dynamically ordering subgoals according to whether their instantiation state gives them uniqueness properties.

Although each of these mechanisms handles only a limited set of the inferences that must be made, taken together they cover a wide range of them. We do not know to what extent these results will extend to other domains, but expect that additional special purpose mechanisms may be required. We hope that these additional mechanisms can also be made domain independent, and that the process of designing new mechanisms will eventually produce a more powerful and complete catalogue of inference mechanisms, that are widely applicable in many domains.

REFERENCES

[Bundy and Sterling 81]
Bundy, A. and Sterling L.S.
Meta-level Inference in Algebra.
Research Paper 164, Dept. of Artificial Intelligence, Edinburgh September, 1981.
Presented at the workshop on logic programming for intelligent systems, Los Angeles, 1981.

[Bundy et al 79]
Bundy, A., Byrd, L., Luger, G., Mellish, C., Milne, R. and Palmer, M.
Solving Mechanics Problems Using Meta-Level Inference.
In Procs of the sixth. IJCAI, Tokyo, 1979.
Also available from Edinburgh as DAI Research Paper No. 112.

[Bundy 73]
Bundy, A.
Doing Arithmetic with diagrams.
In Nilsson, N., editor, proceedings of IJCAI-3, pages pp 130-138. Stanford, 1973.

[Bundy
Bundy, A.
Exploiting the properties of functions to control search.
Research Report 45, Dept. of Artificial Intelligence, Edinburgh, 1977.

[Bundy 78]
Bundy, A.
Similarity Classes.
Working Paper 25, Dept. of Artificial Intelligence, Edinburgh, 1978.

[Dahl 77]
Dahl, V.
Un Systeme Deductif d'Interrogation de Banques de Donnes en Espagnol.
Technical Report, Groupe d'Intelligence Artificielle, University of Marseille-Luminy, 1977.

[Davis et al 77]
Davis, R. and Buchanan, B.G.
Meta-level knowledge: overview and applications.
In Reddy, R., editor, procs of 5th, pages 920-927. IJCAI, 1977

[Gelernter 63]
Gelernter, H.
Realization of a Geometry theorem-proving.
McGraw Hill, 1963, pages 134-52.

[Grosz 77]
Grosz, B.J.
The Representation and Use of Focus in Dialogue Understanding.
Technical Note 151, SRI International, 1977

[Huet 77]
Huet, G.
Confluent reductions: Abstract properties and applications to term rewriting systems.
Rapport de Recherche 250, Laboratoire de Recherche en Informatique et Automatique, IRIA, France, August, 1977.

[Mellish 81]
Mellish, C.S.
Coping with uncertainty: Noun phrase interpretation and early semantic analysis.
PhD thesis, Dept of Artificial Intelligence, University of Edinburgh 1981.

[Nevins 75]
Nevins, A.
Plane Geometry theorem-proving using forward chaining.
Artificial Intelligence 6:pp 1-23, 1975.

[Pereira and Porto 80]
Pereira, L.M. and Porto, A.
Intelligent Backtracking and Sidetracking in Horn Clause Programs implementation.
In Proceedings of the Logic Programming Workshop, Debrecen. , 19(

[Ritchie 76]
Ritchie, G.D.
Problems in Local Semantic Processing.
In Brady, M., editor, Proceedings of the AISB Conference, Edinburgh. AISB, 1976.

[Sidner 79]
Sidner, C.L.
Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse.
PhD thesis, Dept of Electrical Engineering and Computer Science, MIT 1979.

[Wos et al 65]
Wos, L., Robinson, G. and Carson, D.F.
The automatic generation of proofs in the language of Mathematics.
In IFIP Congress 65. IFIP, 1965.